

EFFICIENT STATISTICAL PROGRAMMING? LET'S BE CREATIVE WITH THOSE SAS® STEPS

Keiko I. Powers, Ph.D., Power Information Network, Westlake Village, CA

ABSTRACT

Applied statisticians often rely on SAS for various data processing steps before the actual analyses, and therefore, efficient SAS data management skills are a critical component for being a competent statistician. The less time the statistician spends on data management steps, the more time he/she can spend on statistical analyses, and as a result, it helps the overall project quality improve. Several examples, (1) how to MERGE by character variables, (2) how to sequence the DATA steps and PROC SORT steps, (3) how to use the WEIGHT option for obtaining weighted counts or weighted means, and (4) how and when to use MACRO, illustrate that creative thinking with SAS codes can make the overall performance of the SAS program more efficient. These examples also suggest that willingness to keep exploring various SAS data management routines, including applications of procedures typically used for statistical analysis, will benefit statisticians in improving their data processing skills in the long run.

INTRODUCTION

Efficient SAS programming has been a focus of many papers (e.g., Gilsen, 1999; Lafler, 2000; Powers, 2001; Riba, 1999). In the paper for SUGI 26 (Powers, 2001), I presented several examples in which the same data processing routine was handled more efficiently (i.e., less codes and less complexity) by taking advantage of various SAS procedures and functions. By showing these examples, my attempt was to introduce the idea, "Be open and flexible to new application ideas with the same familiar SAS procedures/functions." The examples in the 2001 paper demonstrated that the simpler and more efficient SAS codes perform the data management task just as well as a set of SAS codes with a more complicated setup. Some of these examples include PROC MEANS to obtain various group-level means, PROC FREQ to produce a SAS data set with the percentage for each group, and PROC SCORE to quickly obtain predicted values using regression parameter estimates. For example, almost all SAS programmers know that PROC FREQ gives the percentage for each category. However, when these SAS programmers are asked to obtain a SAS data set with the percentages or the proportions by category, I have often seen them use a three-step approach where they (1) derive the total count first, (2) merge data containing the total count with the original data, and (3) then divide each category count by the total count to obtain the proportions.

```
/* STEP 1: DERIVE TOTAL COUNT */
DATA VAN2; SET VAN; BY YEAR MONTH MODEL;
IF FIRST.MONTH THEN TOTAL=SALE;
ELSE
TOTAL+SALE;
IF LAST.MONTH;
KEEP YEAR MONTH TOTAL;
```

```
RUN;

/* STEP 2: MERGE DATA AND COMPUTE
PROPORTION */
DATA VAN3; MERGE VAN VAN2; BY YEAR
MONTH;
MSHARE = SALE/TOTAL;
RUN;
```

However, the same proportion data can be obtained by the following PROC FREQ setup:

```
PROC FREQ DATA=VAN NOPRINT; BY YEAR
MONTH;
TABLE MODEL
/ OUT=MSHARE (KEEP = YEAR MONTH
MODEL PERCENT);
RUN;
```

This example clearly illustrates that a simple SAS setup works as well as or sometimes better than a long list of SAS codes to complete a certain task. What these examples imply, then, is how important it is to link a particular data management task with appropriate SAS procedures/functions to complete the task as efficiently as possible without writing many tedious SAS codes. Naturally, not every task can be handled this way – some data management tasks do require, for example, a long list of IF statements or many arrays, or hierarchical and complex do loops. At the same time, many tasks could be handled in a more efficient way. Therefore, it is always beneficial to keep our eyes open to see if there are other easier/simpler ways to perform the data management routine. Going back to the PROC FREQ example, when I asked a few SAS programmers to compute market shares of several automobiles for various US regions, many of them tried the three-step approach I mentioned earlier, or the less efficient SAS setup, despite the fact that the market shares are a mere set of percentages. They did not make the logical connection that the market shares are simply percentages, and therefore PROC FREQ provides the percentage figures they need.

The main purpose of this paper is to provide additional examples to illustrate that, in managing data for various statistical analyses, creative thinking and slight changes in SAS coding can make the data management task more efficient. The following topics will be covered in detail: (1) How to merge data by a character variable and check if the merging step is done correctly, (2) How to avoid many IF statements to create a 'weekly date' variable from a 'daily date' variables (3) How to efficiently create 'weighted

count' data from very large transaction-level data, (4) How to compute weighted means without sorting very large data, and (5) How NOT to write macro routines to achieve more efficiency.

MERGE BY CHARACTER VARIABLES

Data management task:

We have two SAS data sets containing different types of automobile-related information at the region level. We are supposed to merge these two data sets by the region name.

Illustrative solution:

Here is the setup that we are all familiar with for merging two data sets by a variable, in this case, REGION.

```
/* STEP 1: READ IN DATA 1*/
DATA RETAILER;
INFILE IN1;
INPUT NRETAILER 8. MAKE $20. REGION $40.;
PROC SORT DATA=RETAILER; BY REGION; RUN;

/* STEP 2: READ IN DATA 2*/
DATA CONSUMER;
INFILE IN2;
INPUT HHINCOME 8. AGE 8. REGION $40.;
PROC SORT DATA=CONSUMER; BY REGION; RUN;

/* STEP 3: MERGE DATA 1 & 2 WITH REGION*/
DATA ALL; MERGE RETAILER CONSUMER; BY
REGION;
RUN;
```

With this setup, the REGION information from RETAILER is overwritten by REGION from CONSUMER, and therefore it is rather difficult to check if the merging step is carried out without any errors.

Be creative – efficiently check for merging errors:

For a task that involves merging by a long character variable, what I often hear is how we can make sure that these character fields have been input in a standardized way, such as no abbreviation for ' south' or ' north' or a comma versus no comma, etc. Even after data cleaning, sometimes there might be extra spaces between words. The REGION variable is a fairly long character variable with LENGTH = 40, and it might have extra blanks or possibly special characters, such as ' /' or ' .' Naturally, if the names are spelled in various ways in the database from record to record, it would require a complex SAS program to correct the inconsistency problems, or they may have to be corrected by some other means instead. Even when the data are supposed to be cleaned, however, it is highly desirable to incorporate a checking step to make sure that the merging step is done correctly.

Another issue related to merging with a character variable is the length of the character string. Fairly often, we do not have to rely on the entire string of the character variable to perform the merging. Maybe the first ten or fifteen characters of the variable are sufficient to uniquely and correctly merge two data sets. For these reasons, it makes the merging and the subsequent checking much easier if we create a temporary shorter-string variable only to perform merging, as

shown below:

```
/* STEP 1: READ IN DATA 1*/
DATA RETAILER;
INFILE IN1;
INPUT NRETAILER 8. MAKE $20. REGION
$40.;
LENGTH REGION1 $ 40 TEMREG $10;
REGION1 = REGION; /*
create REGION1 specifically for DATA
1 */
TEMREG = SUBSTR(REGION,1,10);
/*create variable for merging here*/
DROP REGION;
PROC SORT DATA=RETAILER; BY TEMREG;
RUN;

/* STEP 2: READ IN DATA 2*/
DATA CONSUMER;
INFILE IN2;
INPUT HHINCOME 8. AGE 8. REGION $40.;
LENGTH REGION2 $ 40 TEMREG $10;
REGION2 = REGION; /* create REGION2
specifically for DATA 2 */
TEMREG = SUBSTR(REGION,1,10);
/*create variable for merging here*/
DROP REGION;
PROC SORT DATA=CONSUMER; BY TEMREG;
RUN;

/* STEP 3: MERGE DATA 1 & 2 */
DATA ALL; MERGE RETAILER CONSUMER; BY
TEMREG;
RUN;

/* STEP 4: CHECK FOR MERGE-RELATED
ERRORS */
PROC FREQ DATA=ALL;
TABLE TEMREG*REGION1*REGION2/LIST
MISSING;
```

For each data set, a temporary variable, TEMREG with the string length of 10, is created for merging. The original variable, REGION, is renamed to REGION1 and REGION2 for the first data set and the second data set, respectively. This way, the original REGION variables from the two data sets are both kept in the merged data ALL in STEP 3. By checking the list produced by PROC FREQ (STEP 4), we can more easily find out if all the merging is done without any problems.

Appendix shows a sample list on how the three variables (frequency counts and percent omitted) will show up under the PROC FREQ step. In particular, if either REGION1 or REGION2 has records with a blank field, those records should be checked for their merging correctness. The data example above shows that the CONSUMER data (DATA2) had records with ' BAKERSFIELD' misspelled, and as the result, the merging step created two records under TEMPREG = BAKERSFIEL and had a blank field under REGION1.

CREATE 'WEEKLY DATE' VARIABLE FROM 'DAILY DATE' VARIABLE

For many of my on-going projects, I have to aggregate daily automobile sales data and create weekly time-series data variables on, e.g., weekly sales volumes, weekly average vehicle price, or weekly average profit margin. The first step for this data management routine is, therefore, to create 'weekly date' variable from the daily sales data.

Data management task:

Create a 'weekly date' variable from daily sales data.

Illustrative solution:

Here is a setup I have seen:

```
DATA AUTO; SET D1;
    DATE=DAY (SALEDATE);
    IF (DATE>=1 AND DATE<=7) THEN
WEEK=1;
    ELSE IF (DATE>=8 AND DATE<=14) THEN
WEEK=2;
    ELSE IF (DATE>=15 AND DATE<=21) THEN
WEEK=3;
    ELSE IF (DATE>=22 AND DATE<=28) THEN
WEEK=4;
```

This setup does generate the weekly date variable, WEEK, that we need to prepare the weekly time series variables, such as weekly vehicle price or weekly profit margin. However, the setup becomes more and more cumbersome as the time period gets longer. In addition, the DAY function returns the day of the month, and therefore, we would have to incorporate the MONTH and YEAR functions for a longer time period. Suppose we are creating weekly time series variables for the past five years. This setup would require 260 IF statements (52 weeks times 5 years). Obviously, this is not the most efficient way to create a weekly date variable from daily sales data.

Be creative – Let's utilize those SAS functions:

Instead of relying on a series of IF statements, if we use the WEEKDAY function, we can generate a weekly date variable for any length of time period with the same setup. The WEEKDAY function assigns a numeric value to the new variable DWEEK, 1 for Sunday, 2 for Monday, and so on, each corresponding to the week day of the sales date.

```
DATA AUTO; SET D1;
DWEEK = WEEKDAY (SALEDATE);
IF DWEEK = 1 THEN DWEEK = 8;
ENDDATE = SALEDAT + (8 - DWEEK);
FORMAT ENDDATE DATE7.;
RUN;
```

With this setup, we can create a weekly date variable that corresponds to the end date of each week. For example, dates between 6/2/03 and 6/8/03 belong to the same week with the 'week-end' date of 6/8/03.

CREATE 'WEIGHTED COUNT' DATA FROM VERY LARGE TRANSACTION-LEVEL DATA

With this task, we are interested in keeping track of the monthly frequency counts for each of the automobile brand, or MAKE (e.g., Ford or Honda). To do so, we have to count the number of transactions for each brand for each month. There is an additional requirement for getting the frequency counts. Since our data are based on a sample, it is necessary to weight the data to adjust for the population size for each brand. We have two data sets; the transaction data consist of several million records, and the weight data contain weights by brand.

Data management task:

Obtain the weighted monthly frequency count for each brand by combining the transaction data and the weight data.

Illustrative solution:

```
/* STEP 1: READ IN TRANSACTION-LEVEL
DATA*/
DATA TRANS; /* the data have several
million transactions */
INFILE IN1;
INPUT MAKE $ MONYEAR;
PROC SORT DATA=TRANS; BY MAKE; RUN;
/*require sorting of several
million records */

/* STEP 2: READ IN WEIGHT DATA*/
DATA WTDATA;
INFILE IN2;
INPUT MAKE $ MKWGT;
PROC SORT DATA=WTDATA; BY MAKE; RUN;

/* STEP 3: MERGE TRANS DATA AND
WEIGHT DATA *
/*require merging of data = TRANS
with several million records */
DATA ALL; MERGE TRANS WTDATA; BY
MAKE;
    WTCOUNT = MKWGT;
RUN;

/* STEP 4: COMPUTE WEIGHTED
TRANSACTION COUNTS */
PROC MEANS DATA=ALL NWAY SUM;
CLASS MAKE MONYEAR;
VAR WTCOUNT;
RUN;
```

This setup has two steps that require extensive computer time and memory space. Sorting the data TRANS with several million records (STEP 1) and merging TRANS with the weight data (STEP 3) will take so much time to execute, or even worse, may not be completed with PC SAS.

Be creative – Obtain the 'unweighted' counts first before merging:

Since the transaction data have several million records, we

should look into some other alternative way to carry out the task. For this particular task, we can first obtain the 'unweighted' count for each brand using PROC FREQ with the OUT option and then multiply these counts by the brand-level weights. The procedure gives the identical set of weighted counts (at STEP 5) as the 'illustrative solution' shown above.

```

/* STEP 1: READ IN TRANSACTION-LEVEL DATA*/
DATA TRANS;
INFILE IN1;
INPUT MAKE $ MONYEAR;
RUN;

/* STEP 2: OBTAIN TRANSACTION COUNTS BY
BRAND */
PROC FREQ DATA=TRANS NOPRINT;
TABLE MAKE*MONYEAR/ OUT=TRANOUT;
RUN;
/*TRANOUT is a much smaller data set */

/* STEP 3: READ IN WEIGHT DATA*/
DATA WTDATA;
INFILE IN2;
INPUT MAKE $ MKWGT;
PROC SORT DATA=WTDATA; BY MAKE; RUN;

/* STEP 4: MERGE TRANSACTION COUNT DATA AND
WEIGHT DATA */
DATA ALL; MERGE TRANOUT WTDATA; BY MAKE;
/* STEP 5: COMPUTE WEIGHTED TRANSACTION
COUNTS */
WTCOUNT = MKWGT*COUNT;
RUN;
/* merging now involves a much smaller data
set TRANOUT */

```

First, the 'unweighted' counts are obtained by PROC FREQ and therefore, the data do not require any sorting. In addition, when the two data sets are merged at the fourth step, the data set TRANOUT containing the unweighted counts is substantially smaller than the original transaction-level data, TRANS. Because of these two computer time saving features, this setup derives the same set of monthly weighted counts for each make (WTCOUNT) much more efficiently than the 'illustrative solution' procedure shown above. Note that the variable COUNT under STEP 5 is a SAS generated variable at STEP 2 under PROC FREQ.

COMPUTE WEIGHTED MEANS WITH VERY LARGE TRANSACTION-LEVEL DATA

Obtaining weighted counts is not the only time that changes in the SAS codes sequence can save the SAS computer time. When computing weighted means, some minor SAS setup changes can drastically reduce the computer time and still produce the same results.

Data management task:

Obtain weighted monthly PRICE means by combining the transaction data and the weight data.

Illustrative solution:

This task has similar problems to the previous assignment, with the very large transaction-level data. We also have to compute weighted means, relying on the weights coming from a separate data set. If we try to perform the merging between the weight data and the transaction data, as shown below, we have problems with the computer time and memory when handling PROC SORT and MERGE.

```

/* STEP 1: READ IN TRANSACTION-LEVEL
DATA*/
DATA TRANS;
INFILE IN1;
INPUT MAKE $ MONYEAR PRICE;
PROC SORT DATA=TRANS; BY MAKE; RUN;
/*require sorting of several million
records */

/* STEP 2: READ IN WEIGHT DATA*/
DATA WTDATA;
INFILE IN2;
INPUT MAKE $ MKWGT;
PROC SORT DATA=WTDATA; BY MAKE; RUN;

/* STEP 3: MERGE TRANS DATA AND
WEIGHT DATA */
DATA ALL; MERGE TRANS WTDATA; BY
MAKE; RUN;
/*require merging of data with
several million records */

/* STEP 4: COMPUTE WEIGHTED MEANS */
PROC MEANS DATA=ALL NWAY;
CLASS MONYEAR;
WEIGHT MKWGT;
VAR PRICE;
RUN;

```

Be creative – Obtain the brand-level means first before merging:

Instead of sorting and merging the transaction-level data, the brand-level means are computed first. By doing so, just as the example on the weighted counts above, we eliminate the two time-consuming steps: sorting and merging an extremely large data set.

```

/* STEP 1: READ IN TRANSACTION-LEVEL
DATA*/
DATA TRANS;
INFILE IN1;
INPUT MAKE $ MONYEAR PRICE;
RUN;

/* STEP 2: OBTAIN MONTHLY MEANS OF
PRICE BY BRAND */
PROC MEANS DATA=TRANS NWAY NOPRINT;
CLASS MAKE MONYEAR;
VAR PRICE;
OUTPUT OUT=TRANMEAN MEAN=;
RUN;
/* TRANMEAN is a much smaller data
set */

```

```

/* STEP 3: READ IN WEIGHT DATA*/
DATA WTDATA;
INFILE IN2;
INPUT MAKE $ MKWGT;
PROC SORT DATA=WTDATA; BY MAKE; RUN;

/* STEP 4: MERGE BRAND-LEVEL MEAN DATA AND
WEIGHT DATA */
DATA ALL; MERGE TRANMEAN WTDATA; BY MAKE;
/* merging now involves a much smaller data
set TRANMEAN */
RUN;

/* STEP 5: COMPUTE WEIGHTED MONTHLY MEANS OF
PRICE */
PROC MEANS DATA=ALL NOPRINT;
CLASS MONYEAR;
WEIGHT MKWGT;
/* with WEIGHT, the means are weighted means
*/
VAR PRICE;
OUTPUT OUT=MONMEAN MEAN=;
RUN;

```

The 'creative' procedure produces the same set of weighted monthly means as the 'illustrative solution' procedure. Because PROC MEANS under STEP 2 uses the CLASS statement, no sorting of TRANS is necessary. The fourth step is merging TRANSMEAN, a substantially smaller data set than TRANS, with the weight data, WTDATA, making the merging step much more efficient with respect to the computer time.

CREATE DUMMY VARIABLES WITH MACRO

MACRO is a great SAS feature that helps our programming be more efficient, in particular for a standardized routine with different data sets or for repetitive SAS tasks. In general, MACRO almost always helps us simplify SAS codes, and as a result, makes it easier to check the SAS log. However, there are some cases where using MACRO does not improve the programming efficiency, and instead simply causes checking the SAS log to be more difficult. This point can be illustrated by showing the case where we have to create many dummy variables.

Data management task:

Create dummy variables for each of vehicle colors.

Illustrative solution:

Here is an example SAS program for MACRO-based dummy variable creation.

```

%MACRO VEHCOLOR(COLOR);

    &COLOR = 0;
    IF EXTCOLOR IN ("&COLOR") THEN &COLOR = 1;

%MEND;
%VEHCOLOR (BLUE)
%VEHCOLOR (GOLD)

```

```

%VEHCOLOR (GREEN)
%VEHCOLOR (RED)
%VEHCOLOR (WHITE)

```

Be practical – sometimes 'no' MACRO makes more sense :

If we examine the above MACRO setup, we see that we do save some programming time by not typing in the same color three times in the dummy variable creation step. However, since we have to type in %VEHCOLOR statement for each color, the saving is not necessarily substantial. If we use a set of IF statements instead, the above example can be set up as follows:

```

BLUE = 0; IF EXTCOLOR IN ("BLUE")
THEN BLUE = 1;
GOLD = 0; IF EXTCOLOR IN ("GOLD")
THEN GOLD = 1;
GREEN = 0; IF EXTCOLOR IN ("GREEN")
THEN GREEN = 1;
RED = 0; IF EXTCOLOR IN ("RED") THEN
RED = 1;
WHITE = 0; IF EXTCOLOR IN ("WHITE")
THEN WHITE = 1;

```

Typing in these codes should not be that time consuming if we simply use 'copy and paste' for each color. Without the MACRO setup, the log statements for the second setup should be much easier to read. This example demonstrates that blindly relying on MACRO without considering and balancing the trade-off between MACRO-induced simplicity (e.g., a shorter SAS program) and MACRO-caused complexity (e.g., hard-to-read LOG statements) sometimes makes the SAS codes less efficient. (In fact, to push the goal even further, more efficient for this task is the usage of PROC TRANSREG, which creates a dummy variable for each level of EXTCOLOR by simply setting a few options.)

CONCLUSIONS

The examples in the present paper demonstrated that sometimes a small change in the SAS programming routine can drastically reduce the computer time or memory. These changes in general did not require any advanced SAS functions or procedures – what I needed was a little bit of creative thinking with the SAS programming logic. Therefore, instead of automatically translating the computation procedure to SAS codes that we are accustomed to, we should always spend extra time to see if there are any alternative more efficient ways to perform the same task. For example, if we have to spend a lot of time spelling out a hundred IF statements, it is possible that there are some other ways to handle the routine more efficiently, using familiar SAS functions or procedures. The examples included in the present study describe the tasks very specific to particular purposes, and therefore its applicability might be rather limited with respect to general data management purposes. At the same time, these examples show that being always critical and creative with

familiar SAS routines helps us discover a way to complete the same data management task more efficiently. This type of mental set, to always search for alternative SAS setups, should help us be a better SAS programmer in the long run.

REFERENCES

Gilsen, B. (1999). SAS Programming Efficiency for Beginners. Proceedings of the Twenty-Fourth Annual SAS Users Group International Conference, Paper 72.

Lafler, K. P. (2000). Efficient SAS Programming Techniques. Proceedings of the Twenty-Fifth Annual SAS Users Group International Conference, Paper 146-25.

Powers, K.I. (2001). Efficient Statistical Programming? - Let SAS Do the Work.. Proceedings of the Twenty-Sixth Annual SAS Users Group International Conference, Paper 80-26.

Riba, S. D. (1999). The DATA Statement: Efficiency Techniques. Proceedings of the Twenty-Fourth Annual SAS Users Group International Conference, Paper 71.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Keiko I. Powers, Ph.D.
Power Information Network
An Affiliate of J. D. Power and Associates
2625 Townsgate Road
Westlake Village, CA 91361
Work Phone: 805-418-8114
Fax: 805-418-8241
E-mail Address: Keiko.Powers@powerinfont.com

APPENDIX

Data format for “Merge by character variables” example

TEMPREG	REGION1	REGION2
CHICO-REDD	CHICO-REDDING	CHICO-REDDING
EUREKA	EUREKA	EUREKA
FRESNO-VIS	FRESNO-VISALIA	FRESNO-VISALIA
MONTEREY-S	MONTEREY-SALINAS	MONTEREY-SALINAS
SACRAMNTO-	SACRAMNTO-STKTON-MODESTO	SACRAMNTO-STKTON-MODESTO
SAN FRANCI	SAN FRANCISCO-OAK-SAN JOSE	SAN FRANCISCO-OAK-SAN JOSE
BAKERSFIEL	BAKERSFIELD	BAKERSFIELD
BAKERSFIEL		BAKERSFIELDS
LOS ANGELE	LOS ANGELES	LOS ANGELES
PALM SPRIN	PALM SPRINGS	PALM SPRINGS
SAN DIEGO	SAN DIEGO	SAN DIEGO
SANTABARBR	SANTABARBRA-SANMAR-SANLUOB	SANTABARBRA-SANMAR-SANLUOB
YUMA-EL CE	YUMA-EL CENTRO	YUMA-EL CENTRO